

Available online at www.sciencedirect.com**ScienceDirect**

Procedia Engineering 131 (2015) 333 – 342

**Procedia
Engineering**www.elsevier.com/locate/procedia

World Conference: TRIZ FUTURE, TF 2011-2014

TRIZ evolution of the Object-Oriented Programming Languages

Victor Berdonosov^a, Alena Zhivotova^b, Tatiana Sycheva^c^a*Komsomolsk-na-Amure State Technical University, Faculty of Computer Technologies, Russia*^b*Komsomolsk-na-Amure State Technical University, Information Systems Department, Russia*^c*Komsomolsk-na-Amure State Technical University, Information Systems Department, Russia*

Abstract

In the series of “TRIZ Future” conferences [1] authors went through the problems of knowledge scope and learning time leading to educational contradiction. The main approach of solving them was the idea of systematic knowledge transferring. TRIZ gives ways to the realizing of the systematization found on the concept of the evolutionary knowledge.

The number of TRIZ specialists (Nikolai Khomenko, Larry Smith, Nikolay Shpakovsky and others) used the evolutionary knowledge in evident or hidden way. Nikolay Shpakovsky (a TRIZ master since 2006), the author of “Trees of evolution”, consequently used the evolution approach [2]. The processes of technical system development are taken up there. TRIZ specialists pay little attention to untechnical systems, especially program ones.

The evolutionary knowledge as an isolated concept is not instrumental. Earlier three parts of evolutionary: “a pattern”, “resources”, “the laws of evolution” were considered to pay special attention on. “The laws of evolution” were supposed to be TRIZ tools (solutions of technical and physical contradictions, SuField transformations, principles and patterns of system development, etc.). These supposals lead to a new concept - TRIZ evolutionary. TRIZ evolution will be self similarity of transformations based on TRIZ tools. In his book Nikolay Shpakovsky analyzes in details “The laws of evolutionary” of the technical system. It is interesting to spread the instrumental approach of the evolutionary knowledge to untechnical systems, too. In the report there was an attempt to apply the TRIZ evolution approach to the program systems, particularly to object-oriented programming languages. The in-depth analysis of the object oriented programming paradigm helped to find out the main contradictions of programming languages of that paradigm. The use of TRIZ tools allowed resolving the contradictions and defining principles, which should be included in a new object-oriented programming language. Such programming languages as Simula-67; Smalltalk; C++; Eiffel; Perl; Python; Php; Java; Delphi; C#; Scala are considered in the report. The principal contradictions, which became “moving force” of each new language, solutions of these contradictions based on TRIZ tools, are defined. Language evolution of programming languages on the basis of “from contradiction to contradiction” is formed. Also the TRIZ-evolutionary map is formed and the development forecast of object-oriented programming languages is suggested.

© 2015 The Authors. Published by Elsevier Ltd. This is an open access article under the CC BY-NC-ND license

(<http://creativecommons.org/licenses/by-nc-nd/4.0/>).

Peer-review under responsibility of the Scientific Committee of TFC 2011, TFC 2012, TFC 2013 and TFC 2014 – GIC

Keywords: TRIZ-evolution, Simula-67, C++, Eiffel, Perl, Python, Php, Python, Java, Delphi, C#; Scala;

1. Introduction

The concept of the TRIZ-evolutionary of knowledge was considered in a number of TRIZ conferences [1] [3] [4]. This concept helps to reveal approaches to resolve the main contradiction of education between the knowledge scope and time of learning. There is a strong connection between the TRIZ- evolutionary and fractal approach in research of objects of different nature. French mathematician Mandelbrot [5] was a founder in the study of fractals. In general, the development (evolution) of the fractal object is in the following way. Source object (pattern) according to the rules of evolution (laws of evolution), using environment resources, repeatedly reproduced (copied), increasing its "complexity". In his works Mandelbrot showed that nature is not completely fractal, but it's fractal to a great extent. Fern is the visual confirmation of that. Nature reproduces in this way both simple (ferns, crystals) and complex objects (animals from ovule to adult).

Similarly to fractal approach, TRIZ-evolutionary approach [5] was suggested to artificial objects. The content of this approach is as follows. There is a new pioneer object (analogue to pattern) for example, a wheel in the form of cut down tree. Then, according to the TRIZ laws (laws of evolution) the object transforms, using resources. In the result, using environment resource, principle of segmentation and the law of enlargement and trimming of systems, the next step the transition to the wheel with spokes is occurred. Similarly, TRIZ-evolutionary approach can be applied to evolution of knowledge [6]. At first, initial regulations - axioms are determined, which equivalent to patterns, for the chosen field of knowledge. Then, resources of the field of knowledge are revealed and evaluated. At last, rules of "construction" are revealed, using TRIZ-tools. Such TRIZ-evolutionary approach was used in the structuring of knowledge in numerical methods [7], CASE systems [8].

In [9] TRIZ-evolutionary of knowledge was considered, which connected with programming, notably paradigms of programming. In principle, evolution of the almost every artificial object can be considered like a "growth of tree". First, there is a germ (mono-element), then it grows into thin twigs (poly- elements), which later developed into the thick branches and, in turn, new thin branches appear on them. This model absolutely conforms to the law of enlargement and trimming of systems [10], notably to the first part is enlargement. After enlargement trimming begins, it was shown on the TRIZ-evolutionary map of programming paradigms. In the article we shall show TRIZ-evolution of the object-oriented programming paradigm, which is the branch of the programming paradigm "tree".

The process of TRIZ-evolution of the object-oriented programming paradigm will consist of:

- description of the source object, i.e. the first object-oriented language;
- identification of contradictions in this language;
- identification of TRIZ-tools, which helps to resolve identified contradictions;
- description of following programming languages, which resolves some contradictions;
- summary for all the most significant object-oriented programming languages;
- construction of the TRIZ-evolutionary map;
- identification of the main properties (mechanisms) that should have more ideal object-oriented programming language.

2. About the object-oriented programming

Nowadays there is no exact definition of the object-oriented programming (OOP) or the object- oriented programming language. In literature, various authors give a different explanation of these terms. Based on these definitions [11] [12] [13], we define the object-oriented programming language as a programming language, basic elements are objects that have their own attributes and methods, and forming a hierarchically organized classes of objects.

According to this definition:

Object is a model (abstraction) of a real essence in a programming system.

Class is an abstract declaration of attributes and methods for a group of similar objects which called instances of class.

Attribute is a parameter declared in a class which characterizes the object (class instance).

Method is declared in a class procedure which defines behaviour of class instances.

In general, the object-oriented approach to development of programs based on four main mechanisms: abstraction, encapsulation, polymorphism and inheritance.

Abstraction is the process of identifying of the essential characteristics of an object that distinguish it from all other kinds of objects and thus providing crisply defined conceptual boundaries, from the viewpoint of the observer.

Encapsulation is the process of compartmentalizing the elements of an abstraction that constitute its structure and behavior.

Polymorphism is the ability of being able to assign a different meaning or usage to something in different contexts and the property of an object respond to a query according to its type.

Inheritance is a mechanism to declare new data types on the basis of existing types in such way that the attributes and methods of the base types become the members of the subtype.

However, equally with development of programs methodology, features of particular programming language are important too, because our constructions must be eventually expressed in some language.

We select the group of mechanisms that define features of program developing on the some particular language:

Syntax is a mechanism of programming language which describes the program structure in form of sets of symbols, words, and operators.

Program structure is a mechanism that determines structure of program code.

Debugging is a mechanism that allows to reduce an amount of errors in the program, and simplify the process of identifying and eliminating errors.

With the development of programming languages, these mechanisms have also evolved and now are sets of items in varying degrees implemented in programming languages. The figure 1 shows the scheme of evolution of these mechanisms with the object-oriented programming languages in which they were used for the first time.

Each next element in a set of mechanisms does not exclude the previous elements; it extends the functionality of the language, increasing its ideality. Further we shall estimate ideality of programming language as an amount of elements of mechanisms that are implemented in the language. We shall also take into account all versions of the language, starting with the mentioned.

Using that information, we shall make a radar chart for each OOP language. Each chart segment conforms to the OOP mechanism. Amount of occupied circles in a segment conforms to amount of mechanisms. Marked circles centers are connected by lines and form some figure. The figure is proportional to the language ideality estimation. The chart shows how language facilities were expanded (tapered) comparing with previous languages.

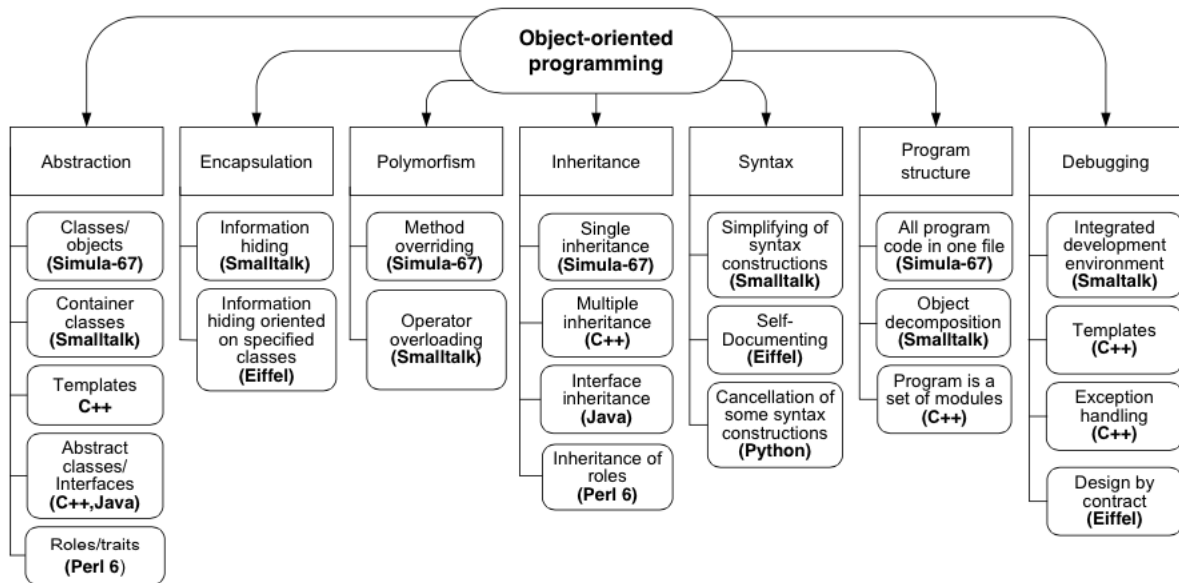


Fig.1 – Evolution of mechanisms of the object-oriented programming

3. Object-oriented programming languages

3.1. Simula-67

Simula-67 was the first object-oriented programming language. Its main ideas and concepts become a fundament of the following object-oriented languages. It means Simula-67 is a “pattern” from which begins the TRIZ-evolution of OOP. Initially, Simula-67 was constructed on the basis of language Algol-

60. Object-orientation and multi-threaded (quasi-parallel) program execution are the most important expansion of Algol-60 that were introduced in the Simula-67. These language resources allow defining program in Simula as a system model that is a set of parallel operating processes. Object in Simula is a program component having attributes and performing operations that are described by action rules [14]. Smula-67 frame is represented on the figure 2.

As any «firstborn» Simula-67 has lot of contradictions. We shall indicate the most significant.

A major shortcoming of the language is a lack of debugging tools and the need to implement a specific algorithm to run a program. Debugging of large applications takes a lot of time because of implementation of the algorithm and searching causes of errors that are not detected by the compiler. Thus, with increasing complexity of developed software time for debugging unacceptably increases (contradiction 1). Also such errors are caused to crash of program. And the greater amount of code, leads to the more probability to miss the error, i.e. with increasing amount of code software reliability unacceptably decreases (contradiction 2).

With the development of computer technology the necessary of writing applications implemented on different hardware platforms has become, but Simula-67 does not solve the problem. With increasing number of hardware platforms efficiency of program unacceptably reduced. (contradiction 3).

The part of contradictions was resolved in Smalltalk programming language with the help of following TRIZ tools. Using the law of transition to a super-system an integrated development environment was established. It has a user interface and provides tools for debugging (*resolution 1.1*).

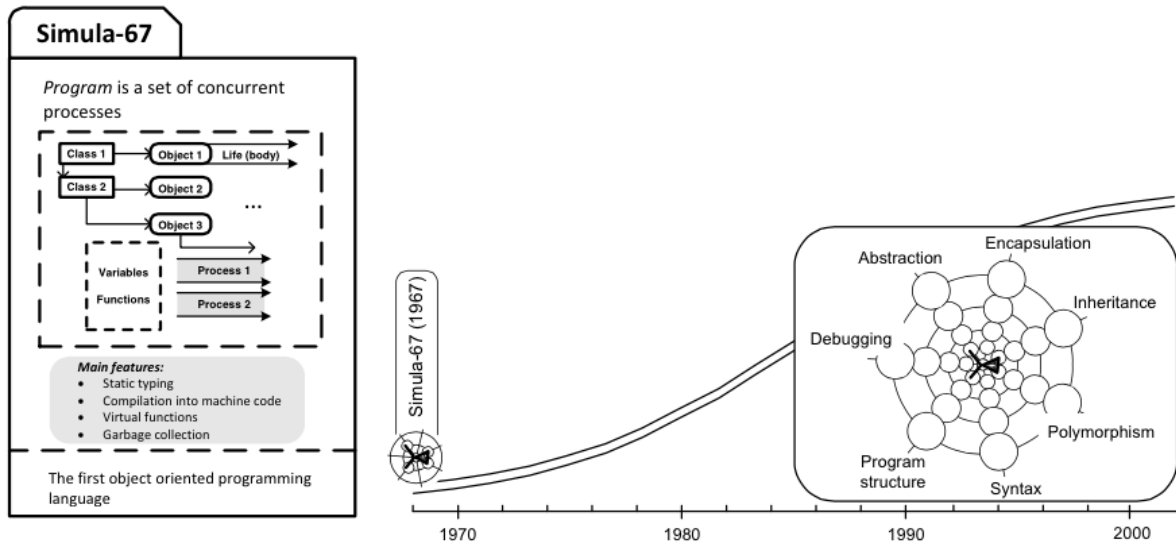


Fig. 2 – (a) The frame of Simula-67 language, (b) location of Simula-67 on the S-curve.

Using the principle of Intermediary the chain of the program compilation was changed: first, program is translated into a bytecode, and then compiled into machine code at the run-time of the program (resolution 3.1). That makes possible to implement on different hardware platforms.

In C++ programming language using the principle of Self-service the possibility of Exception handling was added. The mechanism is used to describe the reaction on exceptions, during run-time of program. (resolution 2.1).

In Eiffel programming language using the principle of Preliminary action, “Design by contract” mechanism was developed, which helps to define formal rules (“contracts”) that are checked at the run-time of program. If one of the contract items is violated, pre-approved and coordinated action comes (resolution 1.2).

Thereby, the first iteration of TRIZ evolution performed. The “rules” of construction are above-listed TRIZ tools.

3.2. Smalltalk-80

Smalltalk programming language (figure 3), developed by Alan Key, brought the popularity to OOP. The main feature of Smalltalk is that all variables are objects, which interact between each other by transmission of messages. Related by inheritance classes are stored in the tree of classes. Herewith any class is the object of a higher degree class. Writing of the program in Smalltalk is a consistent change in the state of its objects [15]. Usage of compilation mechanism into bytecode makes implementation of programs on different hardware platform possible for platforms where Smalltalk virtual machine is supported.

In the same time, Smalltalk has some contradictions too.

It is difficult for specialists with a great experience of programming mastering to learn Smalltalk. In the same time people who don't have a great experience of programming learn the language more quickly. Thus, with decreasing time for learning language by specialists with a little experience, time for learning it by specialists with a great experience in programming in other languages unacceptably increases (*contradiction 4*).

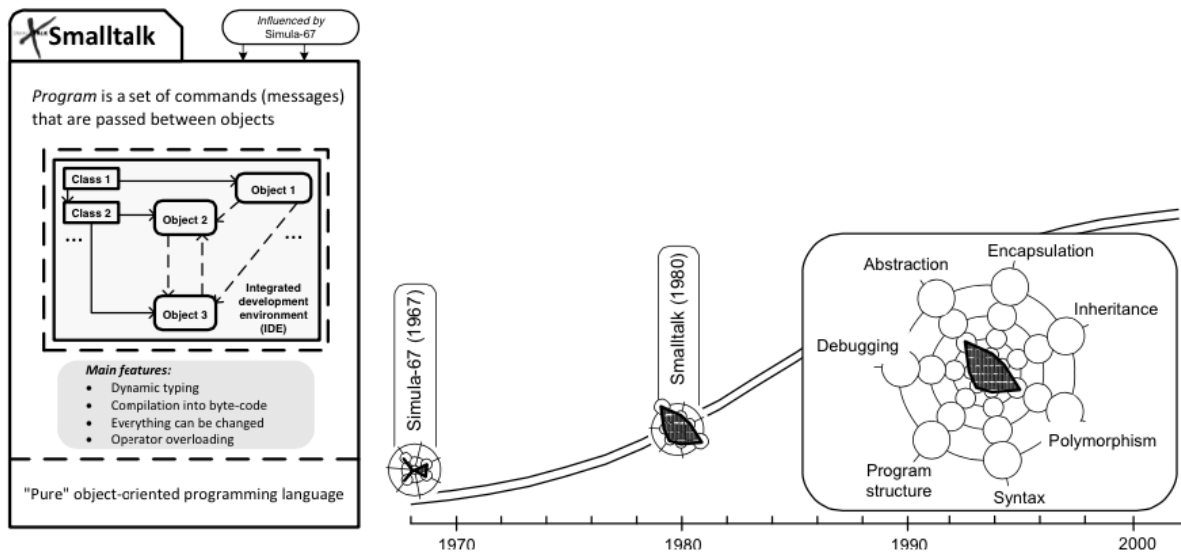


Fig.3 – (a) the frame of Smalltalk language, (b) location of Smalltalk on the S-curve

The contradiction was resolved in C++ by using the principal of Homogeneity. C++ was created on the basis of C programming language (resolution 4.1). Therefore learning of C++ is easy for specialists with a great experience of programming.

Thereby, the next iteration of TRIZ evolution performed.

3.3. C++ and following languages

C++ programming language (figure 4) is one of the most popular object-oriented languages. It was created by B. Stroustrup in 1983 [16]. C++ provides a set of integrated classes and opportunity to declare new types by developers. Classes can inherit one or several classes, providing single and multiply inheritance, correspondingly [17]. In C++ the possibility of describing parameterized classes and functions (templates) and the ability to describe the exception handling in the program were introduced.

Also there are a number of contradictions in C++.

Usage of multiply inheritance produces problems connected with indeterminacy of choice between methods, having same names, of parent classes. For example, we call method Show() for an object and there is no such method in the class, but there are such methods in parent classes. So which one should be provoked? Thus, in condition of usage multiply inheritance, with increasing amount of parent classes indeterminacy of choice between methods, having same names, unacceptably increases (contradiction 5).

In cause of lack of identification between begin/end constructions readability of program code decreases, i.e. with increasing amount of program code readability of the code unacceptably decreases (contradiction 6).

A consequence of the fact, that C++ has never had a standard library and developed for a long time, it was the presence of a large number of libraries to perform the same operations. However, sometimes it's hard to find a good library for some purpose, i.e. with increasing of amount of an external libraries effectiveness of its usage decreases unacceptably (contradiction 7).

In Eiffel language using the principle of Preliminary action, special operators were introduced. They help to declare rules of methods selection when we use multiply inheritance (resolution 5.1) [18].

Defined contradictions of C++ were resolved in Python. Using principles of Self-serving and Preliminary action algorithm of C3 superclass linearization was added to the language. The algorithm sets the order of methods selection when we use multiply inheritance. At the same time, there is an opportunity to redefine this order by developer (resolution 5.2). Using principles of Taking-out and Homogeneity some syntax constructions were expunged, notably the constructions notated begin/end of procedure. Also differentiations of procedures by

alignment of the code for the procedure name were introduced (resolution 6.1). Using the principle of Intermediary new procedure of testing new modules was added, which are offered by developers (resolution 7.1) [19].

In Java using the principle of Partial or Excessive actions, opportunity of interface declaration was added (interface specifies a methods and attributes, which implementing classes should implement). This solution allows obtaining many advantages of multiple inheritance, without full realizing (resolution 5.3).

Further, because of limited lengths of article, resolutions will be described, without formulation of contradictions. Contradictions will be described in the presentation.

In Java using the principle of Taking out lot of unsafe and narrow aimed functions of C++ were excluded. In the same time using the principle of Homogeneity Java inherit syntax from C++, that abbreviate time for master the language (resolution 8.1).

In Delphi, using the principle of Preliminary action virtual method table (VMT) completing before constructor processing. That allows to exclude problems connected with virtual methods calling like C++ (resolution 10.1).

In Perl 6 using the principle of Universality new mechanism – roles (traits) was added. A role allows to model as interface behavior, indicating necessary for the role methods and attributes (specification), as methods implementation inside the role. It helps to use once written code (resolution 11.1).

In C# using principles of Self-serving and Partial or Excessive actions founders tried to hide as much as possible technical details from a program developer, including memory management. Due to that, programmer, using C#, can concentrate on content of problem better than on implementation. In the same time, developer can manage the memory in a mode of “unsafe” code, if it is necessary (resolution 9.1). Also, using the principle of Beforehand cushioning added opportunity to specify the name of specifying interface for methods that have similar names, and to declare different implementations for these methods (resolution 12.1).

Scala programming language using the principle of Homogeneity was build on the Java platform, that provides not only cross-platform, but allows to use modules which written in Java in program developing.

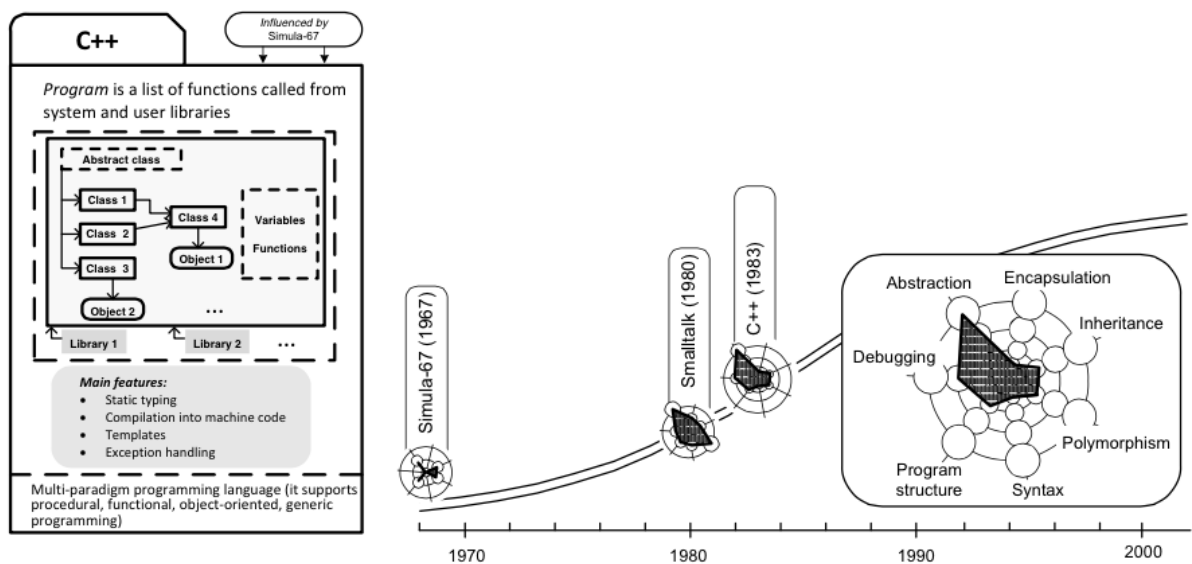


Fig. 4 – (a) The frame of C++ language, (b) location of C++ on the S-curve

Also, Scala has the brevity of Python and Ruby programming languages and supports the functional programming (resolution 13.1).

Uniting all information in the above, there is an opportunity to construct the TRIZ-evolutionary map (figure 5).

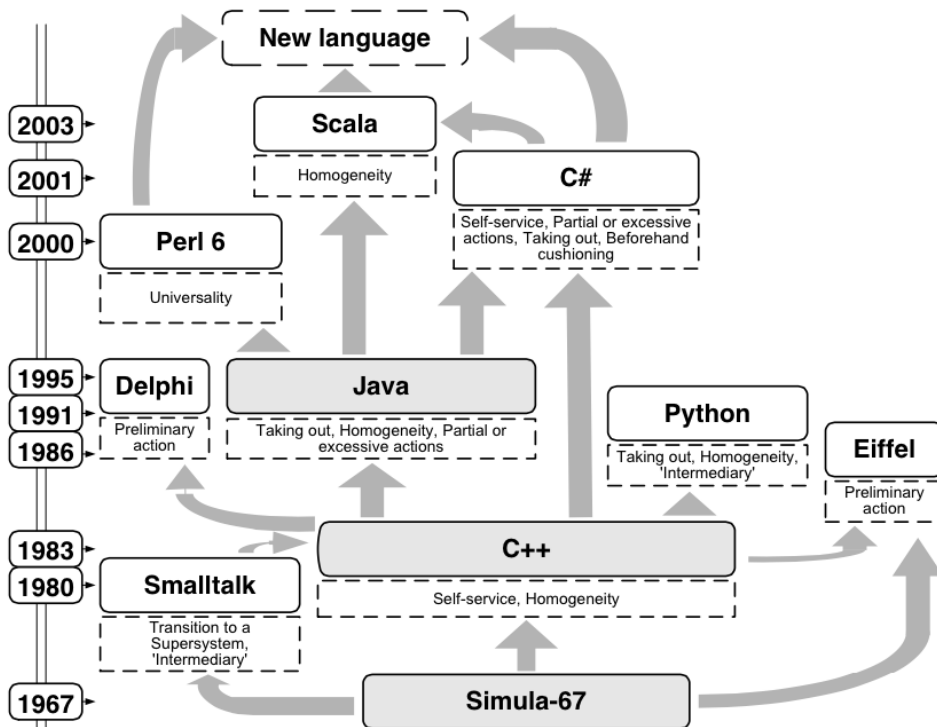


Fig. 5 – TRIZ-evolutionary map of the object-oriented programming languages

4. TRIZ-evolutionary map of object-oriented programming languages

By analyzing the TRIZ-evolutionary map, we can see that languages evolve step by step. Every next stage contains a base language and heir languages, at that there is a language among heir languages, which forms the next stage being base language for the stage. Let's consider these stages.

The first Stage. Base language – Simula-67, heir languages – Smalltalk and C++, which is the base for the next stage.

The second stage. Base language – C++, heir languages – Eiffel (at the same time is the heir of Simula-67), Python, Delphi, Java, which is the base for the next stage.

The third stage. Base language – Java, heir languages – Perl 6, PHP (it wasn't shown on the map, because it didn't resolve any contradictions), C#, Scala.

The fourth stage. At this stage (in our opinion) the mechanism of evolution should be changed. Instead of enlargement that we have seen before process of trimming should be started, i.e. the second part of the law of enlargement and trimming will begin to appear. Partly, trimming has been already started in the Scala programming language. Trimming should be occurred at the level of the mechanisms of object-oriented programming languages. And ideality of object-oriented programming languages should be evaluated by the set of mechanisms, which are included in a programming language. S-curve constructed based on the mechanisms will be as follows.

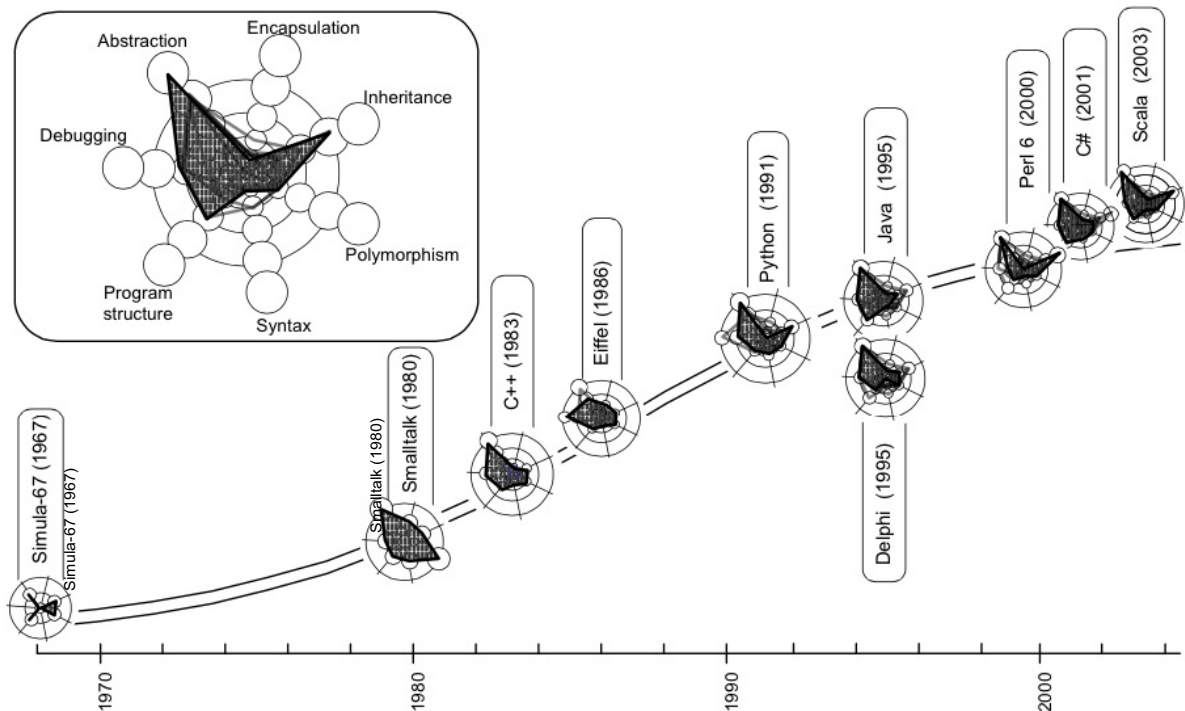


Fig. 6 – S-curve of the evolution of the object-oriented programming languages

Also, usage of TRIZ-evolutionary map helps to considerably increase efficiency of studying by the systematization of knowledge, particularly knowledge in evolution of the object-oriented programming languages. Knowledge systematization realizes in the following way [6]. At first, students learn all TRIZ tools. If there are some reasons and they can't learn all tools, they will learn just resolving contradictions tools. After that, students start study of programming language from the simplest language – Simula-67. They are offered to solve (to program) the simplest task. Then, complexity of the task is increased and again students can solve it. Next they identifying contradictions and try to resolve them, using TRIZ tools.

I.e. they have to offer more ideal programming language or, at least, to describe properties that should be included in that language. If it is necessary the teacher helps the students. Thus, students discover for themselves all following OOP languages. If the course is sufficiently lengthy, students can consider all known modern OOP languages. The most talented students will manage to solve complicated tasks. For example, they can offer new programming language, using the TRIZ-evolutionary approach.

5. Summary

Thus, the analysis of object-oriented programming languages allowed to:

- validate the evolution of the object-oriented programming languages;
- reveal contradictions, which activate the mechanism of evolution;
- describe contradiction resolution TRIZ tools, which became the moving force of the evolution;
- construct the TRIZ-evolutionary map, which considerably helps to intensify the process of these languages studying by students.

It should be note that the progress of the TRIZ-evolutionary map allows to predict ways of evolution of the object-oriented programming languages.

References

- [1] Berdonosov V., “Fractality of knowledge and TRIZ”, Proceedings of the ETRIA TRIZ Future Conference, Kortrijk, 9-11 October 2006, published by CREAX Press, ISBN 90-77071-05-9.
- [2] Shpakovsky N. Trees of evolution. The analysis of engineering information and generation of new ideas. – Moscow: TRIZ- profi, 2006. – 2006 p. ISBN 5-9348-6048-8.
- [3] Berdonosov V., Redkolis E. TRIZ-Fractality of mathematics Proceedings of the ETRIA TRIZ Future Conference, Frankfurt on Main, 6-8 November 2007, published by GmbH press, ISBN 978-3-89958-340-3.
- [4] Berdonosov V., Redkolis E. “TRIZ-fractality of computer-aided software engineering systems”, Proceedings of the TRIZ Future Conference 2010, 3 – 5 November Bergamo-Italy / a cura di Caterina Rizzi. – Printed in Italy by Stamperia Stefanoni. Bergamo University Press, 2010 – P. 153-162.
- [5] Mandelbrot, Benoit, The Fractal Geometry of Nature, Freeman, New York, 1983
- [6] Berdonosov V. Fractality of knowledge and TRIZ [Electronic resource] // ScienceDirect, an inter-national Journal, ISSN 1877-7058, Vol.09, 2011. 752 p., pp 659-664.
- [7] Berdonosov V., Redkolis E. TRIZ-Fractality of mathematics [Electronic resource] // ELSEVIER: ScienceDirect international Journal. – 2011. – Vol. 09. – 752 p. – P. 461-472.
- [8] Berdonosov V., Redkolis E. TRIZ-fractality of computer-aided software engineering systems, [Electronic resource] // ELSEVIER: ScienceDirect international Journal. – 2011. – Vol. 09. – 752 p. – P. 199-213.
- [9] Berdonosov V., Sycheva T. TRIZ-evolution of Programming System Proceedings of the ETRIA TRIZ Future Conference, Dublin, 2-4 November 2011, published by Institute of Technology Tallaght, ISBN 978-0-9551218-2-1.
- [10] Zlotin B., Zusman A., Altshuller G., Philatov V.: 1999, TOOLS OF CLASSICAL TRIZ. Ideation International Inc.
- [11] Booch G. “Object-Oriented Analysis and Design with Applications” (edition 2), Addison-Wesley Professional, 1994, ISBN13: 9780805353402.
- [12] Budd T. “An Introduction to Object Oriented Programming” (edition 2), Addison-Wesley, Reading, Massachusetts, 1997, ISBN-13: 9780201824193.
- [13] Friedman A.L. “Fundamentals of object-oriented software development” (in Russian), Finance and Statistics, 2000, ISBN13: 9785279022878.
- [14] Dahl O.J. “SIMULA 67 common base language”, Norwegian Computing Center, 1968, ISBN:B0007JZ9J6.
- [15] Goldberg A. “Smalltalk: the language and its implementation”, Xerox Palo Alto Research Center, Addison Wesley, 1983, ISBN: 0201113716.
- [16] Stroustrup B. “The Design and Evolution of C++”, Addison-Wesley, 1994, ISBN: 0201543303.
- [17] Sebesta R. “Concepts of programming languages” (edition 9), Addison Wesley Publishing Company, 2009, ISBN13: 9780136073475.
- [18] Meyer B. “Object-Oriented Software Construction” (edition 2), Prentice Hall, 1997, ISBN: 0-13-629155-4.
- [19] Lutz M. “Learning Python: Powerful Object-Oriented Programming” (edition 4), O'Reilly Media, 2009, ISBN13: 9780596158064